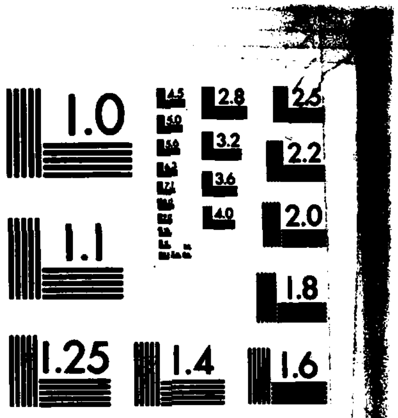
AD-A131 495 FLEXIBLE PARSING(U) CARNEGIE INST OF TECH PITTSBURGH PA 1/1
DEPT OF COMPUTER SCIENCE P J HAYES 22 OCT 82
AFOSR-TR-83-0667 F49620-79-C-0143

UNCLASSIFIED F/G 9/2 NL

END



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS-1963-A

AFOSR-TR- 83-0667

AD A131495

Flexible Parsing: Final [redacted] Technical Report

Philip J. Hayes

Computer Science Department, Carnegie-Mellon University
Pittsburgh, PA 15213

Abstract

When people use language spontaneously, they often do not adhere strictly to commonly accepted standards of grammaticality. The primary objective of this project is to develop flexible computer parsing techniques which can deal with the various kinds of ungrammaticalities that arise, both on the lexical and the phrase level.

The progress towards this goal covered by this report includes:

- The completion of the development and the evaluation of CASPAR and DYPAR, two experimental parsers based on the construction-specific approach to parsing, this approach having been formulated through experience with FlexP, a flexible parser developed earlier under this contract.

- Further development of the construction-specific approach to parsing through the design and construction of MULTIPAR. Like CASPAR and DYPAR, MULTIPAR is based on construction-specific parsing techniques, but aims for much greater linguistic coverage, serving as a vehicle to test whether the construction-specific approach scales up in a more realistic parser. In particular, this work involved development of additional construction-specific parsing strategies and of a control structure through which a large number of such strategies could be coordinated on the parsing of a single input.

- Additional application of the construction-specific approach to flexible parsing to the parsing of an artificial command language in the parser for the COUSIN command interface, a graceful interface for the Unix operating system being developed largely under other funding. This effort represents a parallel track of development and proving ground for the construction-specific approach to parsing.

DTIC FILE COPY

Approved for public release; distribution unlimited.

DTIC
ELECTE
AUG 19 1983
S D
D

UNCLASSIFIED

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER AFOSR-TR- 83-0667	2. GOVT ACCESSION NO. AD-A131495	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) FLEXIBLE PARSING		5. TYPE OF REPORT & PERIOD COVERED FINAL, 1 JUL 81-30 JUN 82
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Philip J. Hayes		8. CONTRACT OR GRANT NUMBER(s) F49620-79-C-0143
9. PERFORMING ORGANIZATION NAME AND ADDRESS Computer Science Department Carnegie-Mellon University Pittsburgh PA 15213		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS PE61102F; 2304/A2
11. CONTROLLING OFFICE NAME AND ADDRESS Mathematical & Information Sciences Directorate Air Force Office of Scientific Research Bolling AFB DC 20332		12. REPORT DATE 22 October 1982
		13. NUMBER OF PAGES 14
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Applied natural language; flexible parsing; bottom-up parsing; friendly interfaces; construction-specific parsing; multi-strategy parsing.

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

When people use language spontaneously, they often do not adhere strictly to commonly accepted standards of grammaticality. The primary objective of this project is to develop flexible computer parsing techniques which can deal with the various kinds of ungrammaticalities that arise, both on the lexical and the phrase level.

The progress towards this goal covered by this report includes:

(1) The completion of the development and the evaluation of (CONTINUED)

DD FORM 1 JAN 73 1473

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

ITEM #20, CONTINUED: CASPAR and DYPAR, two experimental parsers based on the construction-specific approach to parsing, this approach having been formulated through experience with FlexP, a flexible parser developed earlier under this contract.

(2) Further development of the construction-specific approach to parsing through the design and construction of MULTIPAR. Like CASPAR and DYPAR, MULTIPAR is based on construction-specific parsing techniques, but aims for much greater linguistic coverage, serving as a vehicle to test whether the construction-specific approach scales up in a more realistic parser. In particular, this work involved development of additional construction-specific parsing strategies and of a control structure through which a large number of such strategies could be coordinated on the parsing of a single input.

(3) Additional application of the construction-specific approach to flexible parsing to the parsing of an artificial command language in the parser for the COUSIN command interface, a graceful interface for the Unix operating system being developed largely under other funding. This effort represents a parallel track of development and proving ground for the construction-specific approach to parsing.

Accession For
NTIS GRA&I ☒
DTIC TAB ☐
Unannounced ☐
Justification__________

By__________
Distribution/__________
Availability Codes

Dist	Avail and/or Special
A	

1. Research Objectives

1. When people use language spontaneously, they often do not adhere strictly to commonly accepted standards of grammaticality. The primary objective of this project is to develop flexible computer parsing techniques which can deal with the various kinds of ungrammaticalities that arise, both on the lexical and the phrase level. The kinds of ungrammaticality we wish to deal with include at the lexical level:

    - misspelt words
    - novel words whose role can be inferred from context
    - erroneous segmentation between words (arising from the omission of spaces, or the inclusion of spurious spaces or punctuation)
    - lexical items which are entered in one form and then changed to another

    and at the phrase level:

    - input which is broken off and then restarted
    - interjected words and phrases
    - omitted or substituted words and phrases
    - fragmentary or otherwise elliptical input
    - agreement failure
    - idioms

2. The design space for parsers is very large. We aim to develop a set of design choices which will result in parsers well suited to our primary goal. The design choices we are currently using are listed below.

    - bottom-up rather than top-down parsing, except in certain situations in which top-down prediction is highly constraining
    - use of several different parsing strategies, each tailored to a particular type of construction, and selected between on a dynamic basis
    - provision for the suspension and later resumption of a partial parse at a non-adjacent part of the input string

3. We intend to develop flexible parsing techniques in the context of interfaces to interactive computer systems. We are working with two types of interface language:

    a. limited-domain natural languages, i.e. languages with the syntax of (possibly a subset of) natural language, but whose semantics are limited to those of the interactive system being interfaced to.

AIR FORCE OFFICE OF SCIENTIFIC RESEARCH (AFSC)
NOTICE OF TRANSMITTAL TO DTIC
This technical report has been reviewed and is
approved for public release IAW AFR 190-12.
Distribution is unlimited.
MATTHEW J. KERPER
Chief, Technical Information Division

83 08 08 190

b. more restrictive artificial languages of the sort currently found in computer interfaces

Later, we intend to investigate how easily the techniques developed for these kinds of languages can be transferred to more general natural language.

4. We intend to investigate formalisms for specifying domain-dependent grammars in a convenient way for both of the types of language mentioned above.

2. Status of the Research Effort

2.1. Overview

The work covered by this contract from its start in 1979 has involved several distinct phases:

- The initial development of the FlexP flexible parser for restricted domain natural languages, and its evaluation in a gracefully interacting interface to an electronic mail system, covering a period from the start of the contract in July 1979 to early 1981.

- Review of the initial design choices for FlexP in the light of this evaluation, leading to the formulation of the construction-specific approach to parsing, and its preliminary evaluation for applied natural language processing through the experimental parsers CASPAR and DYPAR. This covered the period from early 1981 to the end of the year.

- Further development of the construction-specific approach to parsing through the design and construction of MULTIPAR. Like CASPAR and DYPAR, MULTIPAR is based on construction-specific parsing techniques, but aims for much greater linguistic coverage. The development of MULTIPAR started at the beginning of 1982 and is still continuing.

- Additional application of the construction-specific approach to flexible parsing to the parsing of an artificial command language in the parser for the COUSIN command interface, a graceful interface for the Unix operating system being developed largely under other funding. This effort started in mid-1981, and represents a parallel track of development of the construction-specific ideas mentioned above.

The initial design and development of FlexP and the lessons learned from it leading to the formulation of the construction-specific approach to parsing have been described in the two previous annual reports and so will not be described further here. Even greater detail is contained in the following publications: [2, 3, 1, 4, 5]. This report covers only the period July 1, 1981 to June 30, 1982 and will therefore confine itself to the following topics:

- The completion of the development and the evaluation of CASPAR and DYPAR.

- The design and continuing development of MULTIPAR.

- The design and development of the flexible parser for the COUSIN user-friendly interface.

Separate sections on each of these topics follow.

2.2. Final Development and Evaluation of CASPAR and DYPAR

The final development of both CASPAR and DYPAR was completed in a way faithful to their initial designs which were described in last year's annual report and in greater detail in [2, 3, 1, 4, 5]. To avoid repetition, this report will confine itself to the evaluations and conclusions that were obtained from our experience with these two simple parsers after first reviewing their motivation and design goals.

CASPAR and DYPAR arose out of our evaluation of FlexP which showed that uniform parsing strategies and grammar representations had significant disadvantages for the parsing of ungrammatical input. Because parsing using several different construction-specific strategies was a novel approach, we decided to try out the ideas in two simplified parsers, CASPAR and DYPAR, instead of trying to implement a full-scale parser immediately. CASPAR was designed to show the suitability of construction-specific techniques for ungrammatical input, while DYPAR served as a vehicle to investigate the control problems of coordinating several distinct parsing strategies.

The conclusions obtained from our experience with CASPAR and DYPAR can be summarized as follows:

- The parsing strategy used by CASPAR was tailored directly to imperative case frames. This strategy proved highly successful in dealing with ungrammatical input, much more successful than the uniform strategy employed by FlexP. This result encouraged us to believe that the degree of incisiveness afforded by construction-specific techniques would provide similar advantages across a wide range of constructions.

- Besides performing well on ungrammatical input, CASPAR's construction-specific strategy also in many cases performed more efficiently than the uniform strategy employed by FlexP because of a decrease in the amount of searching necessary. This suggested that a similar gain in efficiency could be obtained through similar techniques for other construction types.

- In the case of both CASPAR and DYPAR, the coordination of the various construction-specific strategies was implemented through the code of the parsers themselves. This would have made it very difficult to add new strategies to the ones already there. A more flexible method for the coordination of multiple strategies is clearly necessary to pursue the concept of multi-strategy construction-specific parsing, since any parser of that type with a wide linguistic coverage would in practice have to build up to a full range of strategies incrementally. Only with very small parsers like CASPAR and DYPAR is it possible to think of all the required strategies in advance and to preprogram their interaction.

- As we have shown elsewhere [4], ambiguities that cannot be resolved by a flexible parser should be resolved in a user-friendly system through a tightly focused interaction with the person who provided the input. Focused interaction requires localized representation of ambiguity, and our experience with CASPAR suggests that this is easier to achieve using a construction-specific rather than a uniform approach. In CASPAR, we analyzed each construction type for the possible types of ambiguity it can give rise to, devised representations for these ambiguity types, and constructed the parsing techniques so that they could recognize each relevant type of ambiguity and generate the appropriate representation. Such localized ambiguity representations would have been impossible to construct with the uniform approach of FlexP.

- To be widely applicable, limited-domain parsers must provide a convenient way to define languages in the class that they recognize. The construction-specific approach offers the advantage that its specialized parsing techniques can operate directly from such domain-oriented language definitions without the need for a time-consuming compilation phase as was necessary with the uniform approach of FlexP. This reasoning rests on the assumption that the most convenient way to express the language from the application point of view is sufficiently close to the "natural" constructions of the language that direct interpretation is possible. Our experience with CASPAR suggests that this assumption is valid.

These conclusions listed here have served as guiding principles in the development of the MULTIPAR parser and of the parser for the COUSIN user-friendly interface as described in the following sections

2.3. MULTIPAR

Given the largely positive experience with CASPAR and DYPAR, we embarked around the beginning of 1982 on the design and implementation of a new parser that we call MULTIPAR, based on the same construction-specific ideas as CASPAR and DYPAR, but incorporating many more construction types. We viewed MULTIPAR as a vehicle for testing whether the construction-specific approach scaled up from the simple pilot parsers already constructed to a parser with adequate coverage for a realistic natural language interface.

DYPAR and CASPAR used two and three different parsing strategies respectively, coordination between these strategies was simple and was "hard-wired" directly into the control structure of the parsers themselves. The much larger number of strategies needed to provide adequate linguistic coverage and the need to make the addition of new strategies easy precluded this "hard-wired" approach for MULTIPAR. The two principal initial objectives in the development of MULTIPAR therefore became:

- The development of construction-specific strategies for a number of additional construction types.

- A control structure which allowed multiple strategies to interact together without their coordination being "hard-wired" into MULTIPAR.

We will describe progress on each of these points separately.

2.3.1. Additional construction-specific techniques

In CASPAR and DYPAR we concentrated on two main kinds of construction: imperative case frames and linear patterns. Clearly, these types fall far short of covering all the constructions common in restricted domain natural languages, so to develop MULTIPAR, it was necessary to identify and devise specific parsing techniques for those constructions commonly found in such languages, but not covered by our existing parsers. Such constructions include: noun groups (with determiners, adjectives, classifiers, and post-nominal cases), declarative and interrogative case frames (in addition to the imperative ones we already handle), wh-questions, relative clause modifiers, conjunction at the noun group and clause level (general use of conjunction may not be necessary in a restricted domain language), and comparatives. We have compiled these construction types as a minimal set necessary for basically habitable restricted-domain natural languages. Other constructions may have to be included later.

For each of these construction types, the following steps are necessary:

- analyze the structure of the construction, paying particular attention to highly restricted, or otherwise easy to identify components;
- devise parsing techniques which take advantage of the structure to parse correct versions of the construction correctly and efficiently;
- extend these techniques to recover robustly and efficiently from situations in which the construction is used incorrectly or ungrammatically wherever such recovery is possible.

In keeping with the construction-specific approach, all this work should be oriented to extracting the maximum possible leverage from characteristics specific to the individual construction types.

By the end of the contract period that is the subject of this report, we had completed these steps for some, but not all of the construction types listed above. The construction types covered during that period include noun groups (with determiners, adjectives, and classifiers), declarative and interrogative case frames, and wh-questions.

2.3.2. Control structure

As noted above, CASPAR and DYPAR used two and three different parsing strategies respectively. Furthermore, coordination between these strategies was simple and was "hard-wired" directly into the control structure of the parsers themselves. The much larger number of strategies needed to provide adequate linguistic coverage and the need to make the addition of new strategies easy precluded this "hard-wired" approach for MULTIPAR. Instead, we required a control structure which allows large numbers of strategies to cooperate on and share information about the parsing of a given input. Based on these considerations and our previous experience with flexible parsing, we established the following goals for the control structure of MULTIPAR:

- Integration of a large number of highly specific and specialized parsing strategies. There may well be several strategies applicable in any given situation.

- Ability to parse bottom-up from the best information available. It is never possible to rely absolutely on any specific piece or feature of a construction being correct.

- As much top-down control as possible. While bottom-up parsing is necessary to form an initial hypothesis about what the structure of an input may be, it is inefficient once that hypothesis has been formed.

- Clean separation between domain semantics and parsing strategies. This is most important because of our intention to apply MULTIPAR to a significant number of different domains.

The following design for MULTIPAR covers the first three of these goals and neither addresses nor contradicts the fourth. The control structure of MULTIPAR involves the following three kinds of object:

- **tasks:** A task represents the goal of recognizing as much as possible of a given subsequence of the input as a certain kind of grammatically specified object (e.g. a task might be to recognize as much as possible between the second and seventh words of "is the price of a display terminal more than a hardcopy terminal" as a <comparable-object>, where <comparable-object> was a grammatical subcategory; remember we are working with restricted-domain language, and therefore, semantic type grammars). Such tasks may specify that the recognition is to be left or right anchored if the whole subsequence cannot be parsed as the desired object. MULTIPAR is driven at the top-level by a task to recognize the whole of an input line as a grammatical super-category, which includes all complete sentences as well as individual objects, and anything else the system being interfaced to is prepared to interpret in isolation. In cases where elliptical replies are expected, the top-level task might be to recognize an object of the type expected.

- **strategies:** A strategy is a method for recognizing a given grammatical constituent. There may be several strategies applicable to any given grammatical category, and a given strategy may apply to more than one type of constituent. Strategies are indexed by grammatical category. Each strategy has a simple initial test based on pattern matching to check applicability to a specific task (i.e. recognizing a given constituent in a given context with possible left or right anchoring), plus a more complicated procedural test of applicability to be applied if the pattern match succeeds. Each strategy has an indication of the amount of grammatical deviation it is designed to cope with, which will correspond roughly to the amount of effort needed to apply it. Strategies may also be limited to left or right anchored recognition.

- **hypotheses:** An hypothesis is the result of applying a specific strategy to a specific task and constitutes the result of the parsing attempt, thus specified. Hypotheses are recorded globally in a blackboard-like structure. Both successful and unsuccessful attempts are thus recorded, and constitute a way of sharing effort between different strategies. The successful ones are analogous to (partial) parse trees.

These three types of structure work together as follows:

1. The top-level task is set up as described above.

2. Given a task, all strategies whose indexing identifies them as suitable for that task are identified, and grouped according to degree of grammatical deviation handled.

3. The strategies are applied in order of ascending ungrammaticality until one succeeds. All strategies for a given level of ungrammaticality are applied (conceptually) in parallel.

4. Application of a strategy means first checking for a precomputed result in the global blackboard of hypotheses, then applying the pattern-match test, then the procedural test, and then if that succeeds, the body of the strategy.

5. The body of a strategy can set up new tasks, and the strategy as a whole succeeds if the sub-tasks succeed.

6. A task succeeds if one or more of its strategies succeed.

To make the preceding description rather more concrete, we present some example strategies, and show how they would function in a parsing some example inputs. The linguistic examples are drawn from the domain of a computer sales assistant. A very simple MULTIPAR strategy is:

StrategyName: comparative-sentence

Recognizes: <complete-sentence>

Pattern: [<be> $X <comparative> $Y]

Body: set up subtasks of recognizing input segments represented by X and Y as <comparable-object>s.

Most of the work in this strategy is done by the simple pattern-matching rule which is its initial test. To see how it might operate consider the input

Is the price of a display terminal more than $100

The strategy would be applicable, and would isolate "the price of a display terminal" as X and "$100" as Y. The two subtasks of parsing X and Y as <comparable-object>s would then be established, with the first being parsed by a strategy which recognized constructions of the "<attribute> of <object>" type, and the second which recognized strings beginning with a dollar sign and followed by digits as sums of money. The strategy would also check that the two quantities were comparable before reporting success, trying coercion at a more flexible stage, and thus making sense of "is a display terminal more than $100".

A more complicated strategy is:

StrategyName: imperative-caseframe

Recognizes: <complete-sentence>

Pattern: [<action-word> $X] (a more flexible version would not be left anchored)

Body: Obtain the case frame of the action word. Scan the input segment represented by X for case markers from that case frame. This divides X up into a number of segments separated by case markers. Set up tasks to recognize objects of the type indicated by the preceding marker for each segment, making allowance for direct and indirect objects.

This second strategy is very similar to the dominant strategy of the CASPAR parser mentioned above. An example input to which it would be applicable is:

replace the display terminal with a teletype

Here "replace" is the action word and "with" is a marker from its case frame. This isolates "the display terminal" and "a teletype" which can be parsed as objects of the appropriate type, in this case <component-set>s.

For an example of flexibility, suppose the case marker "with" is missing, so that the two component phrases cannot be isolated. The strategy then sets up tasks to recognize each of the missing case fillers in the string that it cannot split up. Since the strategies always operate to recognize as much of the given subsequence as possible as the requested category, but will ignore parts that they cannot deal with, the attempt to recognize (in left-anchored mode) a component in "the display terminal a teletype", will recognize "the display terminal", fail to recognize "a teletype", but isolate it, thus leading to its recognition on the second attempt to parse still unrecognized strings as the fillers of unfilled case frame slots.

Of course, there is also no guarantee, given the many roles that individual prepositions fill, that a case marker that is found is really a case marker for the given case frame, as in:

replace the display terminal with a teletype with a paper-tape reader

Here both "with"s are found, leading to two different ways in which the input can be split up for further parsing. The correct reading is finally preferred because it accounts for more of the input, the strong domain constraints making it easy for the parser to refuse to accept "the display terminal with a teletype" as a <component-set>.

Implementation of the above design did not start during the contract period that is the subject of the present report. However, at the time of writing, implementation has begun.

2.4. Design and Development of the Flexible Parser for the COUSIN Interface

At an early stage in the development of the multi-strategy, construction-specific approach to parsing restricted domain natural language, it became apparent to us that a similar approach could be used to parse artificial command languages as well. Accordingly, starting from the beginning of the contract period that is the subject of the present report, we began to develop a flexible parser based on this approach for the COUSIN interface to the Unix operating system, which we are developing under other funding, and which uses an extended version of the standard artificial Unix command language for input. This effort constituted a development track for the construction-specific approach parallel to that represented by CASPAR and DYPAR and their successor, MULTIPAR. The two tracks, however, are not completely independent, since several of the specific techniques developed for CASPAR also turned out to be useful for the COUSIN parser, as the description in the remainder of this section will show. More details of the COUSIN system and its parser can be found in [6].

The command language for COUSIN is the present Unix language, minus the constructions at a level higher than single commands, but supplemented by other language features that make it easier for the user to specify commands. The standard Unix format for command lines is:

<command-name> <options> <arguments>

where <options> is a possibly empty sequence of flags, single characters preceded by dashes, and option markers, also single characters preceded by dashes which identify the next input token as an optional parameter. The <arguments> are a fixed order sequence of parameters to the command that are not identified by any markers, although they may in some cases be optional. An example is:

cc -w -O -o bar foo.c fum.c

which is a call to the C language compiler (cc) with options "w" (suppression of warning diagnostics) and "O" (object code improvement), a flagged option "o" (which writes output to the file named, "bar"), and two arguments foo.c and fum.c, the files to be compiled. Conceptually, cc actually has one argument, the file to be compiled, which may be filled an arbitrary number of times; this type of argument is called a *multiple argument*. A command with two arguments is "cp", which copies a list of files, its first argument, into a directory, its second argument, as in:

cp file1 file2 dir

COUSIN makes two extensions to the standard Unix language: the addition of explicit markers for command arguments as a supplement to the present system of purely positional specification, and the addition of full word flags and markers for options as a supplement to the present system of single characters preceded by dashes. So the above examples could be written for instance as:

cc -O no-warnings foo.c fum.c output-to bar
cc onto dir from file1 file2

When whole-word markers are used, the ordering restrictions of standard Unix are relaxed. Note that this extension makes the language similar in many ways to the kind of language handled by CASPAR - a command verb followed by a set of marked cases. The major differences are that some case markers stand by themselves and have no fillers, and that the Unix positional syntax is still included in the language. This similarity is exploited by some of the flexible parsing techniques described below.

The multi-strategy construction-specific parsing algorithm that we have so far developed for this language is as follows:

1. **Command Identification:** In much the same way as CASPAR finds the verb of its sentence, the COUSIN parser determines which command is being invoked, and locates the syntax description - positional and case information - for the command.

2. **Standard Unix parsing:** Using this syntax information the remaining part of the command line is parsed as though it conformed to the standard Unix syntax for that command, taking only Unix style options and positionally specified arguments into account. If this step is successful, parsing is complete, and no attempt is made to use the case style syntax. This ensures that correct Unix commands which happen by coincidence to use case marker keywords will be recognized correctly.

3. **Extended Unix parsing:** If the standard parse is unsuccessful in any way, the next step is to parse the line according to the extended syntax. The procedure here is the CASPAR case marker scanning algorithm, modified only to deal with case markers with no corresponding case fillers; i.e., a scan is made for any argument marker keywords, or any option keywords, and the arguments and options thus flagged are extracted.

4. **Flexible Unix parsing:** Otherwise, if any of the input string is still not accounted for after this step, a more flexible algorithm is applied. This algorithm is designed to deal with situations in which the user has:

   - used a mixture of marker and positional notation

- misspelt input tokens, either arguments or markers
- used positional notation in the standard Unix style, but has got the arguments out of order
- omitted one or more required arguments
- used standard dash notation with single character flags and markers for options, but has omitted the dash or put the option string other than at the beginning of the input.

Two basic techniques are involved in this flexible style of parsing: scanning for misspelt markers and options, and comparing permutations of the arguments against the input tokens. The first of these is a CASPAR style marker scan, with the possible targets for correct spellings restricted to be markers of the arguments not yet filled. The second technique is specific to the positional style of construction allowed by Unix, and is kept combinatorially tractable by the fact that no Unix command has more than three arguments.

An example will illustrate how this algorithm operates. Suppose, for instance that the user types:

cp onto dir form fil2 file3

when he really intended to type:

cp onto dir from file2 file3

Assume that "dir" is a valid directory name, "file2", "file3" are valid file names, but "onto", "form", and "from" are not valid files or directories. The command cp has two arguments, SOURCE and DESTINATION. SOURCE is a multiple argument of readable files. DESTINATION is an ordinary argument of either a writable directory, or a creatable file (which may or may not already exist). There is an additional restriction that if DESTINATION is a file, SOURCE may contain only one file. The default order is SOURCE DESTINATION.

Standard Unix syntax does not work, so extended Unix syntax is tried. The marker scan comes up with "onto", and "dir" is recognized as a proper DESTINATION, and there are just three remaining arguments which could be assigned to SOURCE, but "form" and "fil2" have failed matches with SOURCE, so extended Unix syntax does not work, and flexible parsing must be tried. Note that if "form" and "fil2" were suitable files for SOURCE, there would have been no need to employ the extra flexibility. The first flexible step is to scan for misspelt markers from left to right. Extended Unix syntax has already accounted for "onto" and "dir", so the scan starts from "form", which is of course corrected to "from". Since "from" is the marker for SOURCE, "fil2" is required to fill the SOURCE argument, and since "file3" satisfies the restrictions for SOURCE, and since SOURCE is a multiple argument, "file3" also is taken into the SOURCE argument. Since "fil2" is required to go into SOURCE the fact that it fails the restrictions on the argument trigger an immediate attempt to spelling correct it. This attempt succeeds, and the parse is correct and complete, without it being necessary to invoke the second permutation phase of flexibility.

The implementation of the flexible parsing algorithm described in this section was completed during the contract period covered by this report, and incorporated into the COUSIN user-friendly interface. The algorithm has proved efficient in the recognition of grammatical input, and robust in its handling of ungrammatical input. In addition, its construction-specific character has made it easy to produce the localized representations of ambiguity in its output which are so important for graceful interaction with the user to resolve the ambiguity (see [4]).

3. Publications

The last of the publications listed below discusses the COUSIN interface project within which the work on flexible parsing for artificial command languages was conducted.

1. Carbonell, J. G. and Hayes, P. J. Dynamic Strategy Selection in Flexible Parsing. Proc. of 19th Annual Meeting of the Assoc. for Comput. Ling., Stanford University, June, 1981, pp. 143-147.

2. Hayes, P. J. and Mouradian, G. V. "Flexible Parsing." *American Journal of Computational Linguistics* 7, 4 (1981), 232-241.

3. Hayes, P. J. and Carbonell, J. G. Multi-Strategy Parsing and its Role in Robust Man-Machine Communication. Carnegie-Mellon University Computer Science Department, May, 1981.

4. Hayes P. J. A Construction Specific Approach to Focused Interaction in Flexible Parsing. Proc. of 19th Annual Meeting of the Assoc. for Comput. Ling., Stanford University, June, 1981, pp. 149-152.

5. Hayes, P. J. and Carbonell, J. G. Multi-Strategy Construction-Specific Parsing for Flexible Data Base Query and Update. Proc. Seventh Int. Jt. Conf. on Artificial Intelligence, Univ. of British Columbia, Vancouver, August, 1981, pp. 432-439.

6. Hayes, P. J. Cooperative Command Interaction through the COUSIN System. Proc. of the Int. Conf. on Man/Machine Systems, University of Manchester Institute of Science and Technology, London, July, 1982.

4. Professional Personnel

1. Philip J. Hayes
   D Sc, 1977, "Some Association-Based Techniques for Lexical Disambiguation by Machine".

2. George V. Mouradian
   M Ph, 1978.

5. Interactions

Ongoing consultation with Dr. Jaime Carbonell, also a faculty member in the Computer Science Department of Carnegie-Mellon University, but not funded under this contract.